計測器向けソフトウェア・プラットフォームの開発

Development of Measurement Software Platform

森成勇一　Yuichi Morinari，加藤康之　Yasuyuki Kato

［要　旨］　昨今，計測器の開発費削減を目的として Embedded Windows や Linux などの汎用 OS（Operating System）を計測器に搭載するケースや他の製品の測定アプリケーションを再利用したいという要求が増加している。その際には，製品の特性に合わせた OS やハードウェアが選択され，他製品のソフトウェアを再利用することが困難になる場合が多い。そこで，製品間での測定アプリケーションの移植性向上を目的として，計測器向けソフトウェア・プラットフォームを開発した。

［Summary］　Recently there has been an increase in cases with generic OS (Operating System), such as Embedded Windows and Linux, and an increase in desire to re-use the other product's application for the purpose of cost reduction. In that case, OS and hardware are selected according to the product's feature, therefore it makes difficult to re-use the software for the other product. Then we developed measurement software platform to improve the portability of measurement application among different products.

1　まえがき

一般にソフトウェア・プラットフォームとは，ソフトウェアやハードウェアを動作させるために必要な OS，ミドルウェア，ハードウェア制御モジュールを指す。このソフトウェア・プラットフォームが提供するインタフェース仕様にのっとったアプリケーション・ソフトウェア（以下，「アプリケーション」という）を設計することで，異なるハードウェア構成においても動作するアプリケーションを開発することが可能となる。

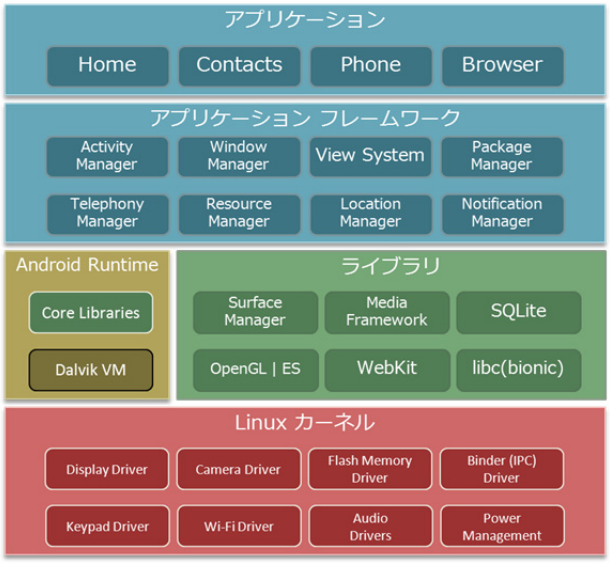

図 1　ソフトウェア・プラットフォーム構造（Android）[1]
Software Platform Architecture (Android)

図 1 にソフトウェア・プラットフォーム構造例として，Android を OS として用いたソフトウェア・プラットフォームを示す。一般的なソフトウェア・プラットフォームの設計では，OS（カーネル部），ライブラリなどの各ソフトウェア階層を抽象化し，階層間のインタフェースを定義する。これらの抽象化した階層間インタフェースを使用して開発されたアプリケーションは，ハードウェア依存性が低くなるため，移植性を高めることができる。

ここでは，ソフトウェア・プラットフォームの考え方を計測器向けの組込みソフトウェアに適用し，異なるハードウェアおよび異なる OS 間におけるアプリケーションの移植性を向上した取り組みの事例を紹介する。

2　開発方針

2.1　開発の背景

近年の急増した有線・無線ネットワーク上の搬送データを測定し，解析するためには，計測器における高速演算処理が必須となっている。加えて急速に機能・性能が進化する CPU やメモリ，ストレージなどのハードウェアにも素早く対応していくことが要求される。当社が扱う計測器開発においても，TTM（Time To Market）の確保やコストダウンの要求に応えるために市販の CPU ボードと Embedded Windows や Linux などの汎用 OS を組み合わせ，製品特性に合わせたソフトウェア・プラットフォームを構築する開発が主流となっている。

また，携帯端末や各種デバイスの開発では，開発の際に PC が

使用されることから計測器にはWindowsとの親和性が要求される。さらに，製造ラインにおいては多数の計測器をネットワークに接続して一斉に制御する場合が多く，計測器には外部からのリモート制御の高速性や長期連続運転における安定性，低価格が要求される。これらの多様なニーズに応え，製品系列に特化したソフトウェア・プラットフォームおよび計測器向けの組込みアプリケーションを開発してきたが，製品系列間での移植性への取り組みについてはまだ不十分である。また，計測器共通の設定や外部制御用のコマンドについても製品系列毎の開発を行っており，開発費削減の阻害要因となっている。

加えて近年ではプロダクトミックスにより，系列の異なる製品間におけるアプリケーションの移植を短期間に効率良く実施したいという要求が増えてきた。この異なる製品系列間でのアプリケーションの移植には，製品系列間でいまだに統一されていないソフトウェア・プラットフォームのインタフェースを統一することが必須である。今回，これらの課題を解決すべく，計測器共通のソフトウェア・プラットフォームを開発した。

2.2 開発の基本方針

計測器向けソフトウェア・プラットフォームを開発するにあたり，製品系列間のアプリケーションの移植性向上を目標とし，以下の3点を基本方針とした。

（1） OSとアプリケーションを分離する

（2） OSは「Windows」ならびに「汎用で安価なOS（Linux）」に対応する

（3） 計測器として共通する設定，コマンドを内包する

3 設計の要点

3.1 基本アーキテクチャ

今回開発した計測器向けソフトウェア・プラットフォームでは，製品系列毎に異なるハードウェアやOSの差分がアプリケーションに影響しないよう，**図2**に示すように，中間レイヤとしてフレームワーク／ミドルウェアレイヤとOS／デバイスドライバレイヤを持たせ，ハードウェアレイヤとアプリケーションレイヤを加えた4層のレイヤ構成としている。

また，OS（Windowsならびに Linux）の違いはフレームワーク／ミドルウェアレイヤで吸収させることでOSの違いによるアプリケーションレイヤ向けインタフェース変更作業が発生しない構成としている。さらに多様な測定ボードへの対応はOS／デバイスドライバレイヤで吸収することによって，ハードウェアやOSと完全分離して独立性を高くし，アプリケーションへの影響を抑える構成とした。

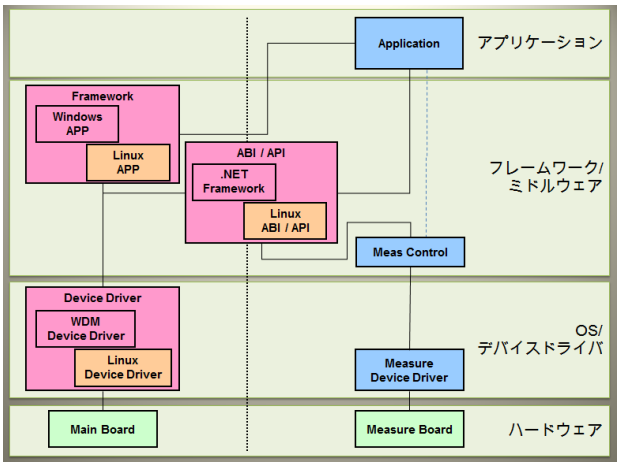

図2 計測器向けソフトウェア・プラットフォーム構成図
Measurement Software Platform Diagram

3.2 Windows版プラットフォームの開発

OEM（Original Equipment Manufacturer）供給可能な組込み向けのWindowsとしては，Windows Embedded Compact（WEC）系とWindows Embedded Standard（WES）系の2種類が一般的である。そこで今回の計測器向けソフトウェア・プラットフォームでは，WEC系と比較してPCとの親和性が高く，社内実績のあるWindows Embedded Standard 2009を採用した。

Windows版プラットフォームの開発にあたっては，基本アーキテクチャのレイヤ構成を満たしている既存製品のソフトウェアを流用し開発に取り組んだ。このベースとなるソフトウェアを流用し，様々な計測器に汎用的に使用できるプラットフォームを構築するためには，ベースとしたソフトウェアの構造検証および共通機能の切り出しを正確に行うことが必須であった。

3.2.1 モジュール間の依存関係の適切な確保

設計書やソースコードによるソフトウェア構造の机上検証は，膨大な時間を要する上に，検証漏れやミスが発生しがちである。そこで，本開発のソフトウェア構造解析では，ベースとしたソフトウェアの依存関係を視覚的・定量的に把握するために，**図3**に示す「依存関係マトリックス」（DSM: Dependency Structure Matrix）という分析技法を用いた。このDSMによってサブシステム，モジュール，ファイル，関数などの構成要素の依存関係（参照度，影響度など）を視覚的に検証した。今回の検証ツールとして用いたLattix（米国Lattix社）というアーキテクチャ解析ツールは，短期間かつ間違いのない検証を可能にした。

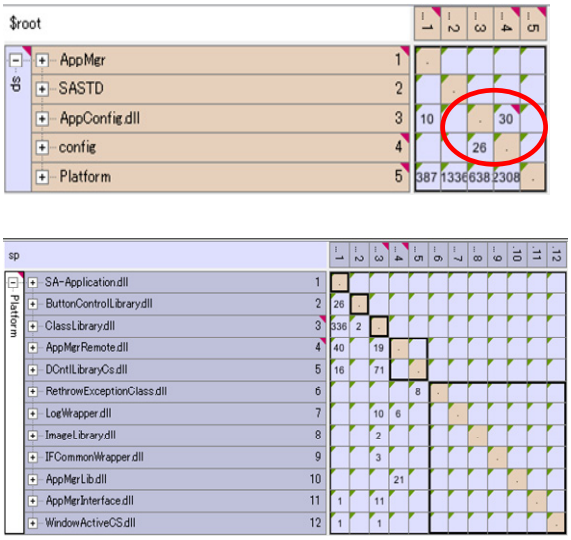

図 3　依存関係マトリクス(DSM)
Dependency Structure Matrix (DSM)

3.2.2　計測器共通機能の搭載

計測器向けソフトウェア・プラットフォームの独立性を維持しつつ，計測器共通の機能を搭載するために，日付・時刻設定，ネットワーク設定，ファイル操作，通電時間計，ライセンスやオプションの管理，GPIB や VXI-11 プロトコル経由での外部からのリモート制御などの計測器一般に求められる機能を定義し，それら以外の個別の製品に特化した機能はベースとするソフトウェアから取り除くことで，**図 4** の赤い枠で囲まれた Platform モジュールの独立性を確保した。そして個別の製品に特化した機能は，**図 4** の Product Customize モジュールとして定義することでカスタマイズ開発が可能な構成とした。

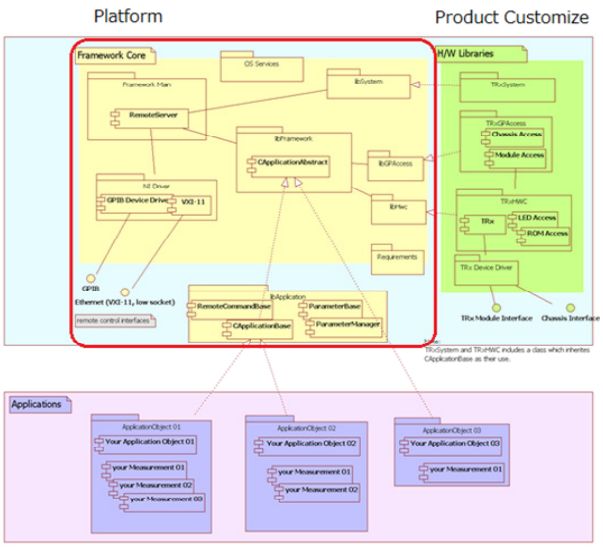

図 4　製品のソフトウェア構成図
Software Diagram of Product

しかし，ここで取り除くこととした機能は，ベースとしたソフトウェアの実装モジュールと強い依存関係があり，ただ単純に不要な機能をモジュールから削除しただけではコンパイル時にエラーが多発し，そのエラーの除去に非常に多くの時間が必要となる。そこで，依存関係にある不要な機能を効率良くモジュールからブロックとして削除するために，前述の Lattix を使用した。この作業においては Lattix によりモジュール間の依存関係を視覚的に確認しながら，モジュール削除時の影響度をシミュレートすることで，コンパイルエラーの大半を抑止することができた。大半のコンパイルエラーを削除した後に残る細かいレベルでの切り離しには，Understand（米国 SciTools 社）というソースコード構造解析ツールを用いて，対象となるメソッドや変数の参照元，参照先，プログラムの制御フロー，構造，クラス継承，関数，変数などを，**図 5** のように視覚的に表現し，1 件毎に確認して切り離す方法を用いた。

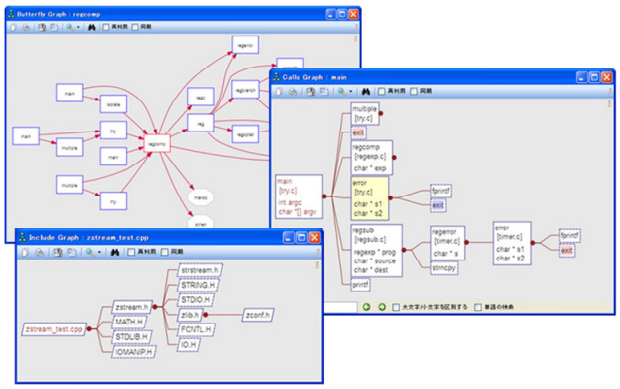

図 5　バタフライ図，関数呼び出し図，インクルード図
Butterfly Graph, Call Graph, Include Graph

3.3　Linux 版プラットフォームの開発

計測器向けソフトウェア・プラットフォームでは，最新のハードウェア・ミドルウェアを低コストで開発・量産するために，プラットフォームの OS として Embedded Windows の他に Linux も採用している。組込み用途を謳う Linux ディストリビューションは商用製品を含めて数多くあるが，長期稼働における信頼性を考慮しなければならない計測器向けソフトウェア・プラットフォームとして本開発では，サーバ系オープンソース・ディストリビューションを採用した。

計測器向けソフトウェア・プラットフォームに Linux を採用するにあたっては 3 つの課題があった。1 つ目は，輸出規制に抵触するリスクがあること，2 つ目は計測器特有の GPIB インタフェースを実装する必要があること，3 つ目は Windows 版と Linux 版のアーキテクチャを共通化することである。

3.3.1 輸出規制対策

オープンソースのソフトウェアを利用する上では，輸出管理上の制約が発生することがある。ソフトウェアにおいては暗号技術が規制にかかることが多く，今回採用した Linux のディストリビューションにおいても輸出規制に抵触するリスクがあった。そこで，カーネルやライブラリから暗号技術を削除することで，輸出規制に抵触しないディストリビューションを独自に作成した。このオリジナルのディストリビューションを使用することで，プラットフォームの利用者が，暗号技術の有無を選択して実行環境を構築することができるようにしている。

3.3.2 Linux 版 GPIB の搭載

計測器では外部インタフェースとして GPIB が必須となる。この GPIB 制御のために，Linux 版計測器向けソフトウェア・プラットフォームのデバイスドライバ，ミドルウェア環境ではオープンソース実装されている linux-gpib パッケージを採用した。このパッケージは，多くのハードウェアで使用実績があり，既存アプリケーションの移植が容易な業界標準の API（Application Programming Interface）を持っているため，計測器向けソフトウェア・プラットフォームが想定するハードウェアに対しても短期間に移植することができ，開発コストを大幅に削減できる。

3.3.3 Windows 版と Linux 版の移植性向上

計測器用の組込みアプリケーションの移植性を最大限に高くするには，アプリケーションをプラットフォームの API のみで構成して開発すれば良い。しかし，Windows 上で動作する既存のアプリケーションを Linux 環境に移植する場合，プラットフォームの API だけでなく.NET Framework の API をコールしている部分があり，プラットフォームに Mono（米国 Xamarin 社）などの互換ライブラリを使用して置き換えるか C++の STL（Standard Template Library）を使用してコードを置き換えることが必要となる。

Linux 版計測器向けソフトウェア・プラットフォームでは，使用実績のない Third Party のライブラリで問題が発生した場合に改修できなくなるリスクと，今後の計測器向けソフトウェア・プラットフォームとして計測器特有の拡張を行う可能性を考慮して，C++の STL を採用した。この C++の STL を採用することによって.NET Framework の特徴である Boxing の対応が必要となる。Boxing を使用している箇所は，修正箇所が多岐に渡るため，Boxing に相当する専用のデータクラスをあらたに設計することで対応した。また，クラス設計では実際に使用する全ての型を網羅した上で，プラットフォームの API をコールしている部分のなかで特に使用頻度の高い代入 operator などの置換作業を効率的に実施できるようにクラスの実装を行った。

また，.NET Framework の特徴であるメモリ自動解放についての注意も必要である。Linux 版計測器向けソフトウェア・プラットフォームのフレームワークでは STL の採用により意識してメモリを解放しなければならなくなった。このメモリの解放漏れを防ぐため，ソフトウェア検証ツールとして C++test（米国 Parasoft 社）による静的解析およびフロー解析を使用してメモリの解放忘れ（メモリリーク）発生箇所を検出し対策した。しかし，ツールによるメモリリーク発生の検出箇所の指摘には誤検出（メモリの取得と解放を異なるモジュールで行うなど）も多く，指摘内容について 1 件毎に目視によるソースコード・レビューが必要であった。

3.3.4 出荷後のアップデート対応

Embedded Windows も Linux も，ソフトウェア・コンポーネントは OS イメージ構築時に選択できるが，開発中および開発完了後にコンポーネントのアップデートが必要となる場合がある。これらのアップデートに対応するため，OS 毎に必要となるパッケージセットを標準構成として定義し，容易にアップデートが可能な機能を盛り込んだ。

3.4 Remote Studio 開発によるリモートコマンド開発の簡略化

計測器向けソフトウェア・プラットフォームでは，当社の主要な計測器に実装されている IEEE 488 および SCPI（Standard Commands for Programmable Instruments）規格に準拠した外部制御コマンド（リモートコマンド）機能も提供している。リモートコマンドは数万種類にも及ぶため，従来の設計ではコマンド仕様作成・プログラミング・テスト・取扱説明書作成といった一連の設計プロセスにおいて，検討項目漏れや担当者間の連絡ミスや記入ミスが発生するという問題が発生していた。

従来のリモートコマンドの設計プロセスを分析した結果，冗長な繰り返し作業や，定型作業が多いことがわかった。そこで，コマンド定義をインプットとして，仕様書，ソースコード，テストケース，取扱説明書を自動生成するツール（Remote Studio）を開発し，設計における単純なミスの混入を防いだ。

Remote Studio では，コマンド仕様書や取扱説明書の入力フォーマットを所定の様式で記述することで，従来手作業で行なっていたコマンド仕様書から取扱説明書などへの変換作業や整形作業を自動化している。また，Remote Studio より出力するソースコードならびに，それに対応するテストケースは，初期値，上下限

値，プレフィックス，サフィックスなどの単体テスト範囲をカバーし，独自のテストケースを追加可能な仕組みとした。

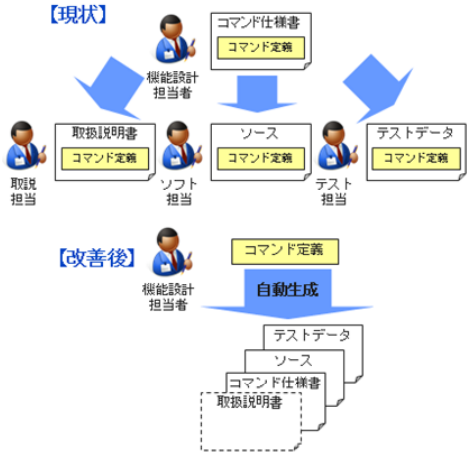

図 6　リモートコマンド開発プロセス
Process of Remote Command Development

Remote Studio は，計測器向けソフトウェア・プラットフォームにおいて提供する IEEE 488 および SCPI 規格に準拠したコマンド解釈仕様に基づくソースコードやテストケースを自動生成しているため，ソフトウェアの生産性向上のみならず仕様解釈齟齬の解消といった品質面における効果も大きい。

4　むすび

今回は計測器向けソフトウェア・プラットフォームとして，Windows 版と Linux 版を開発した。この開発によってアプリケーションを OS や処理系から独立することが可能なソフトウェア構造を構築した。しかし，プログラムの実装では OS や処理系に依存する部分がまだ残っている。今後の展望として，測定ボード制御の共通化，アプリケーションとのインタフェースの共通化，OS のシステム・コールの隠蔽化，また Remote Studio においては，リモート制御テスト完全自動化に向けて，機器内部で自動テストできる仕組み，登録したコマンドデータをヘルプ・ガイダンスのデータとして使用できる仕組み，データベース化によりコマンドデータを他機種に展開できる仕組みなどの拡張を図っていきたい。

参考文献

1）　Android Platform Security Architecture, http://source.android.com/tech/security/#android-platform-security-architecture

執筆者

森 成 勇 一
アンリツエンジニアリング㈱
統合エンジニアリング推進部

加 藤 康 之
アンリツエンジニアリング㈱
統合エンジニアリング推進部

公知